10/042,028 PTO-892

(12) UK Patent Application (19) GB (11) 2 385 158 (13) A

(43) Date of A Publication 13.08.2003

(21) Application No 0202848.8

(22) Date of Filing 07.02.2002

(71) Applicant(s)
Guang Yang
Flat 7, 23 Rosevale Street, GLASGOW, G11 6EL, United Kingdom

(72) Inventor(s)
Guang Yang

(74) Agent and/or Address for Service
Guang Yang
Flat 7, 23 Rosevale Street, GLASGOW, G11 6EL, United Kingdom

(51) INT CL[7]
G06F 17/30

(52) UK CL (Edition V)
G4A AUDB

(56) Documents Cited
EP 1122692 A2
US 6282539 B1
US 5539903 A
US 20010052908 A

(58) Field of Search
UK CL (Edition V) G4A
INT CL[7] G06F
Other: Online: WPI, EPODOC, JAPIO, THE INTERNET

(54) Abstract Title
A system for inserting hierarchical data into an existing document

(57) The invention relates to a system for inserting hierarchical data into an existing target document. The system retrieves data from database or other data sources and puts the data into a multi dimensional data repository. The system creates a document model based on scanning the target document and employs directives embedded in the document as hints to create sub document models that are embedded into their super models. The document model may be a hierarchical model containing string constants, variable expressions and sub-model. The system recursively scans and parses the elements of the document model. It then inserts the retrieved data into the appropriate elements of the document model. The same data repository can be shared by different documents for different presentations.

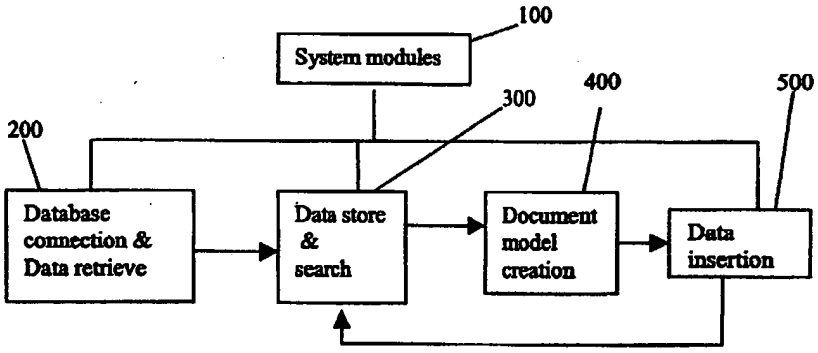

FIG. 1 illustrates the modules of the system according to the invention

GB 2 385 158 A

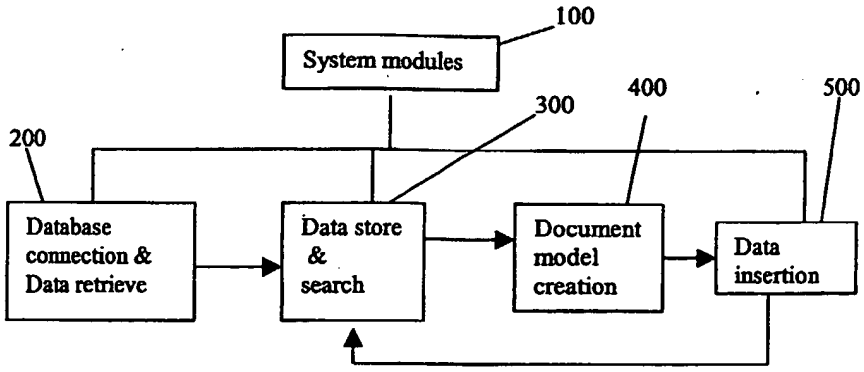

FIG. 1 illustrates the modules of the system according to the invention

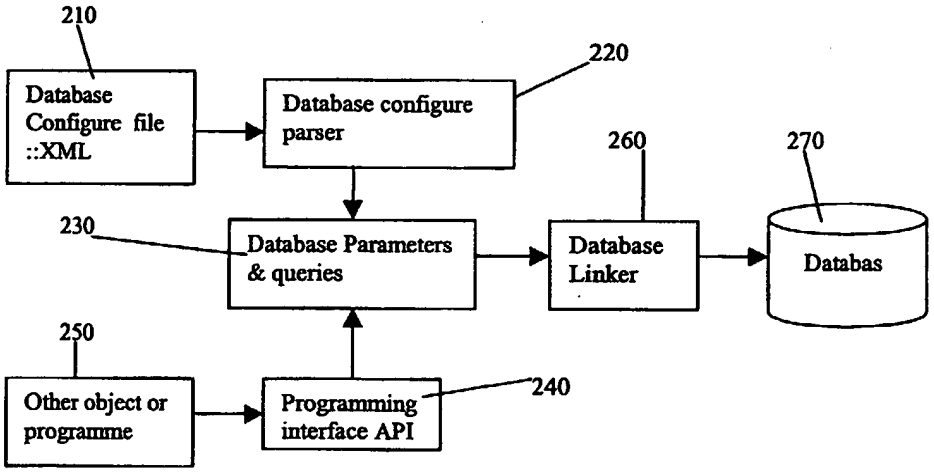

FIG. 2 illustrates database connection and data retrieve module

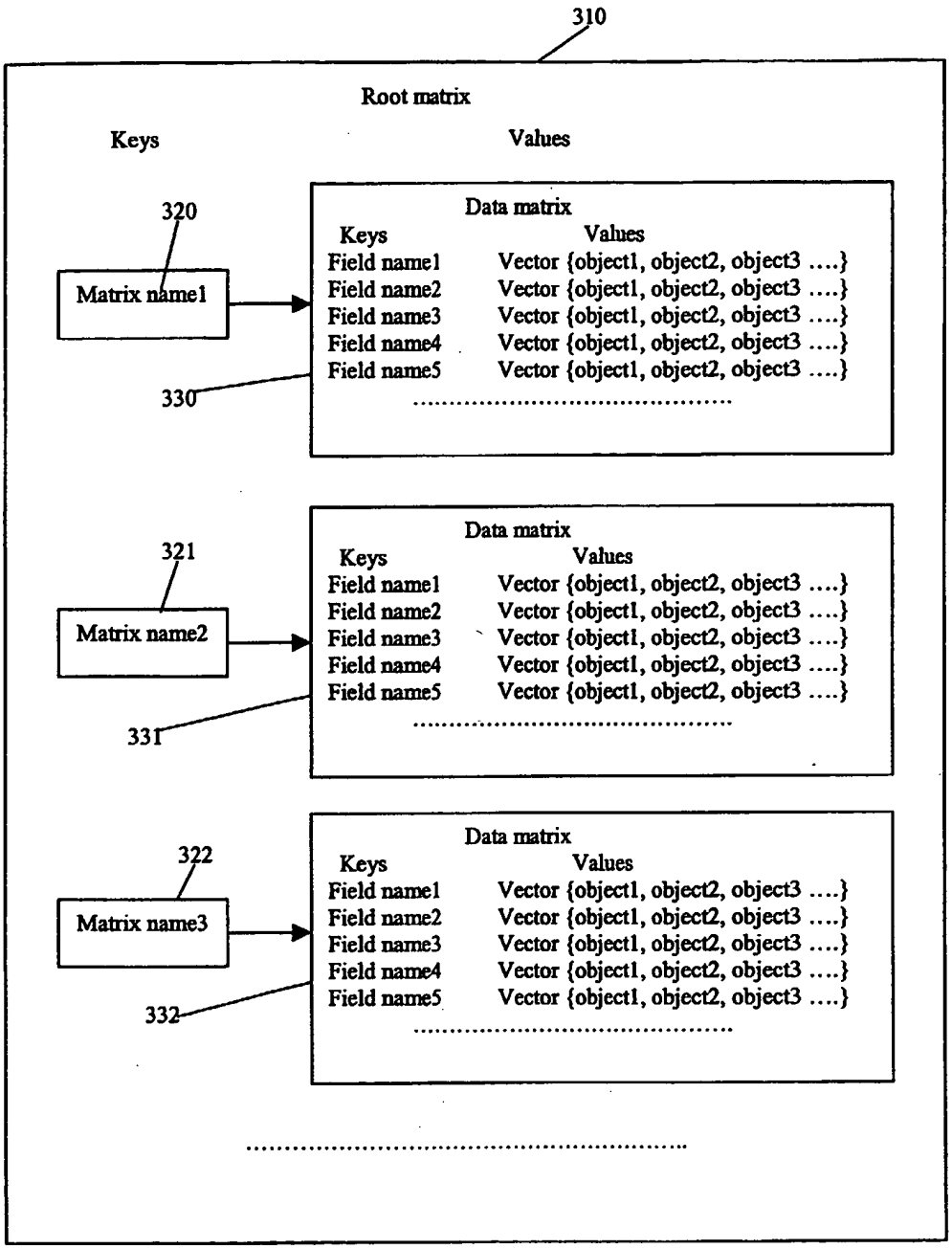

FIG. 3 illustrates the data structure model

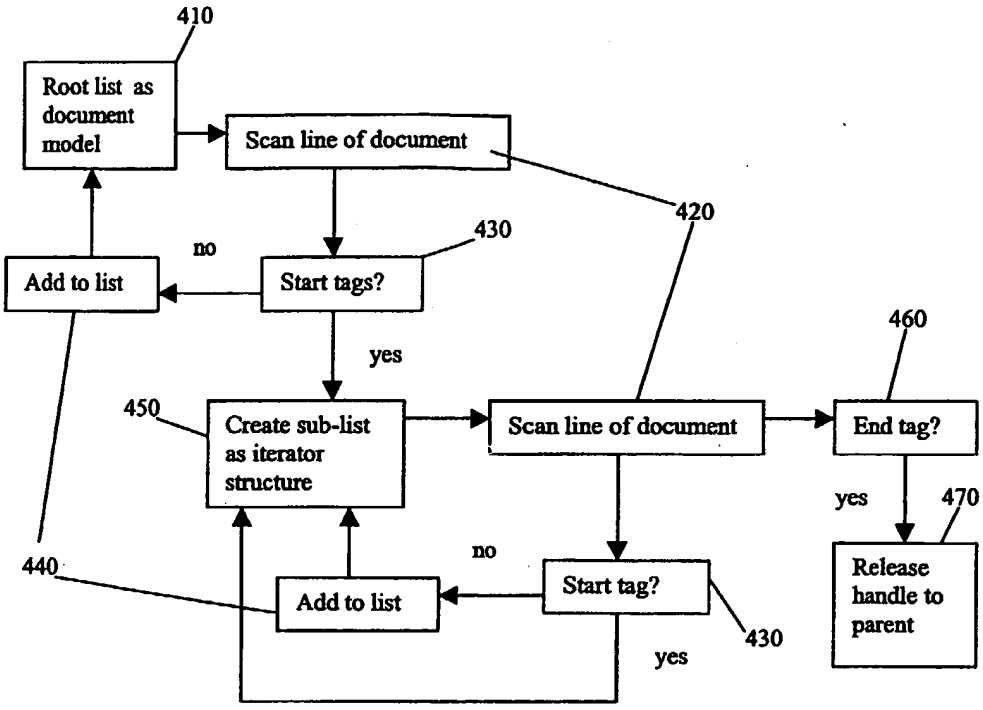

FIG. 4 illustrates the creation of document model

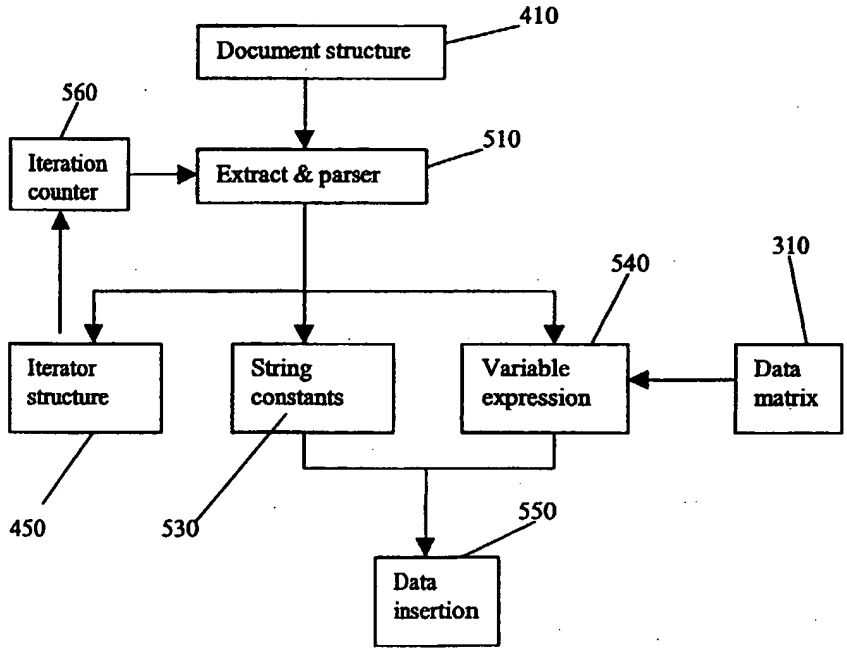

FIG. 5 illustrates the procedure of data insertion

2385158

A System for Inserting Hierarchical Data into an Existed Document

The field of this invention relates to computer software improvement and implementation for inserting data into an existed document or file, and is particularly directed to automatic insertion of hierarchical or nested data into an existed document in accordance with directives input by an user. The system recursively renders the elements of the document model with data retrieved from a data repository using the regular search expression. The data repository and regular search expression are defined by the system.

Background of this invention is to present a large corpus of hierarchical data in different types of documents with an universal system data repository. Different from the process of which is a creation of new documents, the invention is about to insert the data into a target document and reserve all features of the document. It allows users to employ various tools to create the target document for the data insertion. It generates a structured document in terms of flexibility, efficiency and customisation.

Elements of a document are normally string constants or in texture format so that the document contents to be printed or published over web and other network. The elements of a structured document, such as catalogue, are usually arranged in nested or hierarchical manner. Whereas, the data stored in database are normally in the terms of relations either in a table or plain atomic object storage manner. Most commercial applications store data in a relation database where the data are stored in a set of two-dimensional relation tables. The data in the database are normally to be presented in document elements or the portions of document elements for publication or print. The presentation of data is a process mapping the data to appropriate document elements. Mapping structured data to document elements appears to be a challenge, especially when the data has large corpus. It would be difficult and time consuming process.

To present database data in the terms of hierarchical document elements, a computer program called report generator is to map the data into the desired document elements and to generate a documental report. Many commercial database products, like MS Access, Oracle, have report generator to present data as a report document. However, these report generators generate report documents based on pre-defined document templates for specific applications. In generally, different applications may use different templates in different report generator. Some of well defined templates may be shared by a group of applications, but they are still not universal. Moreover, the report generator generates a new document. Users may need additionally programming or manually operation to copy the report elements to the portion of an existed document to accomplish the insertion. It would be not efficient.

The current commercial solution of universal is to employ the Extensible Mark UP Language XML. The report generator produces a XML document containing the database data. The solution uses a XML parser to parse the XML document and additional programmes are still needed to interact with the parsed document elements. The additional programmes have to comply with specific presentations of applications. For instance, XSL or XSLT programming to each HTML, DHTML, XML, PDF, Latex presentations individually. In brief, XML provides universal data model which may be used by different presentations but the XML is an indirect way

for the process of data insertion and still requires the additional operations to accomplish the insertion. Both XML and report generator seem not to solve the problem of unlimited iteration for customised insertion.

Because of the complexity and size of hierarchy data, as well as their cumbersome usage requirements, considerable efforts is expended by an user when trying to present data in a portion of a document while other portions of the document keep unchanged. For example, an user wants to create a HTML web document containing a large corpus of hierarchical data as well as with multiple links of images and URLs. A great deal of effort has to be expended to write a programme for production of such web document from scratch. This invention provides a means for the user to insert the data into the specified portion of the document while other parts of the document keep unchanged, which reserves all links and features of the document. Therefore the user can use a HTML editor to create a nice looking and rich linked web page as a prototype. Then insert the hierarchical data into the prototype. In similar manner, the user can create other document, like XML, PDF, and so on, with the same data repository.

More specifically, the objectives of the invention is trying to enable an user to retrieves data from database and then to:

1. insert the data into various types of documents without additional programming specified by applications, which improves the system flexibility;
2. insert the data into different portions of a document, the user can insert a single value of data, a group of values of data, iterated and nested values of data into the specified portions of the document, while keep other parts unchanged to reserve the useful features of the document. It can produce a rich and customised data presentation, because it allows the user to use other tools to create the document prototype;
3. an universal data repository for different presentations or documents.

The terminology used in this description is explained as following sections in accordance with this invention.

A data repository is a run-time or persistent container to stored data and is an implementation of a data model. In this description, the data repository is the generalisation of the data matrices object. The data model is a way how to store data in a data matrices object. A data matrices object is a container that holds a group of data matrix objects. Each data matrix object within the data matrices is a two dimensional container to contains data fields. The structure of a data matrix is similar to a table, so a data fields as a table column. Each data fields have field names and a set of field values. An atomic data is sometimes called cell value which locates a specified field vector.

The variable expression is a simple regular expression indicating the system where to retrieve the correspond value of the variable via its name and position. A variable name is a data field name of a data matrix. The variable expression likes, not restricts to, variable hint or sign, matrix name, variable name, and value position. The formula likes: $table_name.variable_name.row_number.

A document model is a structure of a document for insertion. The directives of document directions may be represented by a set of iteration tags or other document operation tags. The iteration tags are pairs of beginning and end tags that instruct the system to create sub-model within its super model during document model creation and to perform recursive iteration during the data insertion process. The iteration tags may contain the data matrix names and handle of the iterations, therefore the variable expression within the iteration tags could be a variable name like $Field_name. The formula may be, not restrict to, <iterator name=table_name:start_index:end_index> like <iterator name=table_name:2:15> means it will 13 iterations from 2 to 15. The start and end indices specify the start and end positions of the iteration and indicates the number of loops. If the indices are missing, like <iterator name=table_name> which means to iterate whole table.

Unlike traditional report generator, this invention inserts data retrieved from database into a portion of an existing target document using variable expression and directives such as iteration tags. It is that the essential features of this invention, which can be described in more detail as the followings:

1. The system scans the target document to create a scanned-in document model which hold scanned contents.
2. The system uses iteration tags as hints to create sub-document model to hold sub-contents for data iterating. The iteration tags usually appears in pairs, one is start iteration tag, another is end iteration tag. The iteration tags instruct the system to create the sub-model and put scanned content elements in the sub-model until the end iteration tag. It can insert other pairs of start and end iteration tags between exiting pair tags to perform nested iterations. To support nested iteration, a recursive algorithm was implemented to repeat the processing, if the system find other iteration tags between the current pair of tags. It allows user can created as many iteration as it likes.
3. The scanned-in document model contains string constants, variable expressions which may embedded in the string constants, and sub-document models which may contains the same types of objects as its super model.
4. The system recursively iterates the elements of the document model and parses variable expressions that may be contained in the elements. The system retrieves data from the data repository using the variable expression as paths and substitutes the variable expressions with the data to accomplish the insertion

The system completely separates the document model with the data model. The both are independent of each other. It is that make the document model agnostic about data model it dealt with to increase flexibility. An user can create different document models that can communicating with the data model as long as their variable expressions are consistent. For instance, the user can create Latex, PDF, HTML document models and insert data into these model using the same variable expressions complying with the data model.

According to the running procedure, the system contains 4 modules: a database connection and data retrieve module, a data storage and search module, a document model creation module, and a data insertion module.

The database connection and data retrieve module employs the industrial standard database connections, such DAO, ODBC and other connectivity, to connect a

database and use standard query to retrieve data from either a relational or object-oriented database. The connection can be both local connection and network connection including client/server and web connections. Alternatively, the module may use or integrate other applications to retrieve data via the programming interface. In this description, a local connection with XML configuration is given as an instance of the implementation. The XML configure file to provide the parameters for the database connection and data query. The retrieved data will be put into the data storage and search module.

The data storage and search module is a process to create a data repository in compliance with the system defined data model. The data model is a compound data structure which may contain sub-structures. The system defined a root container called the data matrices. The data matrices object comprises of many other sub-structures called data matrix, and each data matrix has an unique name as an associated key within the data matrices. A data matrix in turn contains many data field objects. A data field is made of a field name and a set of corresponding field values called vector. Therefore the data matrices is 3 dimensional storage and search facility. The first dimension is the names of data matrix objects, second dimension is field names, and third is an indices of field values in the value set. The system can search an atomic data in the data matrices by using the 3 dimensional parameters as a search path. In practice, an user can implement the data model based on the above concept using various data structure, such as data array, vector, hash table and so on. The data matrices object can either be persistent store as a file or in computer memory.

The document model creation and data insertion module is the basis of essential features of this invention described before. It creates a document model from scanning a target document and composes the document model with the data repository to accomplish the insertion of data to the target document.

A preferred embodiment of the invention will now be described with reference to the accompanying drawings in which:
FIGURE. 1 illustrates the modules of the system according to the invention.
FIGURE. 2 illustrates database connection and data retrieve module.
FIGURE. 3 illustrates the data structure model.
FIGURE. 4 illustrates the creation of document structure.
FIGURE. 5 illustrates the procedure of data insertion.

FIGURE 1 illustrates an overall structure of the system module 100. The functionality of the system 100 is to retrieve data from a data repository such as a database and another data source and insert these data into a portion of a target document using customised directives as hints or as data passages. The system 100 can be both standalone application and integrated into other applications. It could be either a local or network application. The system contains 4 modules: database connection and data retrieve module 200, data storage and search module 300, document model creation module 400, and data insertion module 500. The 4 modules are independent of each other and communicate with the predefined interfaces.

The module 200 is activated first and connects the system with a database or another data source. The module will execute query to retrieve data from the database. The retrieved data will input into module 300. The module 300 will stored the data in the

manner that can be easily searched by regular expression. Then the module 400 scans a target document and creates a hierarchical document model which contains the information about the document structure for the insertion. The module 500 recursively iterates the elements of the document model and inserts data retrieved from the data repository into the document model to produce output.

FIGURE. 2 illustrates database connection and data retrieve module 200. The module described here is based on relation database connection via database connectivity, like ODBC, DAO and so on. The module has two types of inputs, a script file called database configure file 210 or a programme input 250.

The script file 210 is a texture file which embeds the database connection parameters and queries. For instance, if the script file is a XML file, it looks like:

```
<?xml version='1.0' encoding='utf-8'?>
 <DBML>
  <Connector>
    <DBDriver>$Database driver </DBDriver>
    <URLStub>$Database stub</URLStub>
    <DBName>$Database name</DBName>

    <DBURL>$Database stub + $Database name  </DBURL>
    <UserName>$User name</UserName>;
    <Password>$User password</Password>
    <Host>$Machine host name</Host>
    <Port> $Communication port </Port>
 </Connector>
<pool>$pool size </pool>
 <Query name= "$table name">$Quary 1</Query>
 </DBML>
```

Here $ indicates that it should be replaced with its real value. The XML file is used to configure the database connection and provide the SQL query script.

A programme called Database configure parser 220 parses the script file into elements and store the elements into a data structure. In the case of XML file the parser based on the DOM model of XML parses the xml file and stores the results of these elements of the xml file in a ad-hoc way, i.e. stores in a hash-table with associated key related to the element's tags or tag attributes. The storage is the database parameter and query object 230 to hold these parameters and SQL statements resulting from the parsed xml file. The storage object 230 is a separate layer between input and database linker 260. It can also be the interface of input methods and provide a flexible mechanism to accommodate different input methods 210 and 250. The method allows an user to change its query or connection to increase flexibility.

Alternatively, an user can write its own programme to retrieve data from database or integrate the module into another application via the programme interface API 240. The retrieved data are also imported into the object 230.

The database linker object 260 read the database connector parameters from the database parameters and query 230 to make database 270 connection. The necessary parameters required by database 270 mainly include the database driver, database

URL, user identification and password. If the connection is successful, the database linker 260 can execute SQL statements from the database parameter and query 230 and retrieve data from the database 270. The resulting data will be stored in a data matrices 300.

FIGURE. 3 illustrates the data structure model which makes of the data storage and search module 300. The retrieved data are stored in data matrices object 310 that is a compound data structure to contain other sub-structures. Each sub-structure may have its own sub-structures. It builds so-called 3 dimensional or layers data structure, i. e. it is a nested data structure in compliance with the regular search expression. The basic or root dimension is the root matrix 310. It is a data structure stores the data matrix objects, 330, 331, 332, and so on (33x), their associated keys are 320, 321, 322 (32x), respectively. This data structure can be implemented as an instance of hash tables or maps that stores the both objects and their associated keys. The keys will help the system to retrieve the objects from the data structure. The data matrices object 310 also has behaviour for the system to manipulate the stored objects. It can add, delete and retrieve a object with its associated key. It provides the first entry for the system to search a data matrix object with its associated key.

The second dimension is a data matrix 330x and the data matrix is also a data structures similar to data matrices object 310. The data matrix stores data field objects and their associated keys. It can manipulate the stored data field objects and provide second search entry to find a set of field values with the field name. They are also more likely to be an instance of hash table or maps.

Third dimension is data field objects that are stored in data matrix 33x. A data field is a container of a set of data filed values or cell data. The data field values are atomic data of the system. These atomic data are stored in a data structure such as linked data array or vector, which keeps the order of the objects during storage. The data structure allows identical data objects to be stored in its different portions. Every stored atomic data object has its index and the index services as search entry to find it specified atomic data. Generally, an index search is faster than key search.

The first and second dimensions used keys to search the corresponding objects. In the first dimension, a key is the name of a data matrix, so in second dimension, the key is the name of a data field. These names have to be unique within their container, more specifically, the names of data matrix 32x have to be unique within a data matrices 310, the names of data fields have to unique within a data matrix. Third dimension uses an index to search its corresponding object. Thus it provides the facility of name space search. The search expression is **matrix_name.field_name.index**. All manipulations of a data object have to follow the procedure indicated by the expression. The data matrices is generally described as a data repository defined by the system.

The default values for missing of dimensional parameters of search expressions are in following rules. If an index of a field value is messing, it should return all values in the specified data field. If a field name 32x is missing, it should return all fields in the specified data matrix. If a name of a data matrix is missing, it should return null due to the fact that no specified data can be found.

FIG. 4 illustrates the module 400 of the creation of document model using recursive programming on scanned-in document. An user writes directives as hints on a target document to indicate the system where to insert the specified data objects into the documents. The hints includes variable expressions complying with search expression and directives specifying iterations. There are two kinds of insertions: atomic data insertion and hierarchical data insertion. The former is to insert atomic data objects into the specified portion of the target document. The expression of the hint may be $matrix_name.field_name.index, while sign $ indicates that it is a variable. An user could use another special character as the variable indication. The later is to insert hierarchic data objects into the target document indicated by the directives in the document. These directives, in the description, are represented by iteration tags. The user has to write start tag to start iteration and end tag to end iteration and also the number of iteration. The tags looks like,

<iterator query=matrix_name:bigin_index:end_index>

The default value means all data, for instance, <iterator query=matrix_name> means iterating all rows in specified data field; <iterator query=matrix_name:x> iterating from x to end; <iterator query=matrix_name:x:y> iterating from x to y. The field name within the iterator likes $field_name, since the iterator handles indices and the name of the specified matrix object. The end tag might be </iterator>. The user can define his own pair of iteration tags if he wishes. An example for Latex document is give:

```
\section{\sffamily{Course Entries}}
{\it Primary Course:} $course.Course.1
<iterator query=department>
\subsection{$Department}
\lhead[$Department]{}
\rhead[]{$Department}
<iterator query=course>
\foreignlanguage{nohyphenation}{\subsubsection{$Course \index{courseindex}{$Cour
se {\it \\($Department)}}}}
{\it Credits:} $Credits \hfill {\it Level:} $Level
{\it When Taught:} $Taught
{\it Timetable:} $Timetable
{\it Requirements of entry:} $EntryReq
{\it Co-requisites:} $CoReq
{\it Excluded Courses:} $Exclude
{\it Assessment:} $Assessment
{\it Degree Examination taken in:} $DegreeExam
{\it Resit Examination taken in:} $ResitExam
{\it Aims:} $Aims
{\it Honours Course Prescription:} $Honours
{\it Course Co-ordinator:} $Coordinator
</iterator>
</iterator>
```

The procedure of creation of the document model is described as follows:

1. The system first creates a root list 410 to hold all data. The root list is a generic data structure, normally a linked data array or a vector that stores data elements in sequence.
2. The system scans the target document 420 and parses the string line.

3. If the string line doesn't contain an iteration start tag 430, it adds the string 440 into the corresponding list, it might be root list or sub-list depends on the iteration status. The root list 410 indicates the outmost iteration.
4. If the string line contains a start iteration tag 430, it creates a sub-list 450 which is similar data structure as root list. Repeat the procedure 2, and 3. There after the string will be added to sub-list 450 until an end iteration tag is met. The sub-list will be added to its super list.
5. When the string contains an end iteration tag 460, the system would release the handle 470 to its super list, which is to transfer the current iteration to its outer iteration. The scanned string after the end iteration tag will be added to the super list thereafter.
6. For nested iteration, the producers 2, 3, and 4 are performed repeatedly according to the application.

The list stores the structured information of the document and can be described as in the terms of document model. The root list is a document model and the sub-lists are sub-models. The sub-models are also called as iterator structure in the case of processing description.

A recursive programming technique is employed to implement the procedure. It provides elegant and flexible solution for users to carry out unlimited iteration theoretically. Any computer programming Language, which supports recursive feature, such as C++, Java, can implement the procedure. Here is example in Java code fragment:

```
public void scanLine(String s, Vector v0)  throws IOException
{
    if(s == null)
     s = BReader1.readLine(); //skip empty line
    else if(s.trim().equals(getIteratorTag()))
    {
     Vector v = new Vector();
     String con = new String();
     do
     {
       String s1 = BReader1.readLine();
       if(s1.trim().equals(getIteratorTag()))
          scanLine(s1, v); //recursive loop
       else if(!s1.trim().equals("</iterator>"))
            v.addElement(s1);
       con = s1;
     }while(!con.trim().equals("</iterator>"));
          v0.addElement(v);
    }else
           v0.addElement(s);
}
```

The created document model contains string constants, variables expressions indicated by $ character or other user defined character, and sub-models which in turn may contain the same types of contents as their super model to form the hierarchical structure. The variables present by prefix character and variable expression are regular expressions to indicate the system where to find their corresponding values in the data repository. Each sub-model contains iterator information including the indices of start

and end iteration, super model information, and the name of the data matrix it belongs.

FIGURE. 5 illustrates the module 500 of data insertion. Data insertion is a processing that recursively transverses the elements of the document model 410 and inserts data retrieved from the data matrices 310 into the appropriate document elements. The procedure includes following steps:

1. It extracts document elements from the document model and parses these elements. The parsed result is one of the three types, iterator structures 450, string constants and variables.
2. If the element is iterator structure 450, set up an iteration counter to start index and go back step 1, repeat it until the iteration counter reach the end index of the iteration.
3. If the parsed element is a string constant 530, the element will be directly written into the target document using data insertion 550.
4. If the parsed element is a string with variable expression 540. The system will parse the expression onto dimensional search entries including matrix name, field name and data index, then use these entries to fetch data from data matrices 310. Parse the expression is firstly to check whether the variable is outside an iteration loop. If it is, the parser tokenises the expression and builds a data array containing these entries. If it is not outside the iteration loop, the parser has to get the matrix name and iterator counter 560 as data index from the corresponding iterator information.
5. The fetched data from the data repository 310 will be inserted into the appropriate elements of the target document by substitution of the variable expressions.

The following sections will give an example to show how the invention works.

Suppose an university wants to publish its course catalogue on the web in HTML document. The course catalogue is organised as the way that the university has many departments, and each department teaches many courses. Different department teaches different courses. The departmental and course data are stored in department and course tables separately in a relation database. The name of department table is "department", and the name of course table is "course". It requires to publish these courses grouped by the departments. The web publisher, first all, writes a script file in XML format for database connection and query, shown as:

```
<?xml version='1.0' encoding='utf-8'?>
<DBML>
 <Connector>
        <DBDriver>sun.jdbc.odbc.JdbcOdbcDriver</DBDriver>
        <URLStub>jdbc:odbc:</URLStub>
        <DBName>diary</DBName>
        <DBURL>jdbc:odbc:diary</DBURL>
         <UserName>user_id</UserName>
         <Password>pwd</Password>
         <Port>:708</Port>
          <Host>localhost</Host>

        <PoolSize>50</PoolSize>
</Connector>

<deparment>select * from department</deparment>
 <course>select * from  course</course>
```

```
</DBML>
```

Then he creates a HTML prototyping file which has nice looking and multiple links using commercial editor tool. The HTML document is used as the target document for data insertion. The web publisher write the hints on the target document to instruct the system where and how to insert data as:

```
Other HTML elements
.....
<!- -data inert here - ->
<iterator query=department>
<h4>Department: $Department</h4>
<iterator query=course>
<br><b> $Code</b> <img src="http://www.gla.ac.uk/graphics/x.gif" border="0" alt
="" width="50" height="1"> <b> $Name </b><br>
<i>Credits:</i> $Credits <img src="http://www.gla.ac.uk/graphics/x.gif" border=
"0" alt="" width="150" height="1"> <i>Level:</i> $Level <br>
<i>When taught:</i> $Taught<br>
<i>Timetable:</i> $Timetable <br>
<i>Requirements of entry:</i> $EntryReq<br>
<i>Co-requisites:</i> $CoReq<br>
<i>Excluded Courses:</i> $Exclude<br>
<i>Assessment:</i> $Assessment <br>
<i>Degree Examinations taken in:</i> $DegreeExam <br>
<i>Resit Examinations taken in:</i> $ResitExam <br>
<i>Aims:</i> $Aims <br>
<i>Honours Course Prescription:</i> $Honours <br>
<i>Course Co-ordinator:</i> $Coordinator<br>
</iterator>
</iterator>
.....
```

The system was implemented in Java, JDK1.4, the latest version which has the XML parser and linked hash map classes. It could be implemented in other programming languages that have recursive programming features. Once running the system, the system connects to a database and retrieves data from the database using the database parameters provided by the database connection script file.

The retrieved data are imported into a data repository called "data matrices". The root matrix called data matrices object is an instance of HashMap class. The data matrices can stored the name of a data matrix as key object and the data matrix as value object. It can retrieve the stored object via its key. The second dimensional object called data matrix is the instance of LinkedHashMap. The structure is similar to above except it keeps a sequence of stored object. A data matrix stores data field names as keys and data field values as value objects. The third dimensional object is data field values which is an instance of Vector. The data field stores atomic object of this application. The atomic object is used to insert to a target document and substitute its variables. All manipulations have to via the three dimensional entry comply with the regular expression for the system to retrieve data.

Then the system scans the target document by calling the scanLine subroutine. It creates a data vector to hold the scanned result. When scan to department iterator, it creates a department vector to hold departmental data. All scanned data afterward will be stored in the department vector until the end iteration tag. Similarly, it scans the

course iterator, it creates a course vector to hold course data. The course vector is an element of department vector and the department is an element of data vector. The data insertion process is the iteration of the data vector. When the iteration encounters the department vector, it starts department iteration. After it gets department name variables, it starts the course iteration. Once the course iteration finished, it releases the handle to another department iteration for next run, and so on. During the iterations, the system parses and substitutes the variable expressions with the data retrieved from the data repository. Therefore the hierarchic data are inserted into the target document.

The course catalogue in HTML formatting is ready for publish. The process only insert data into the related portion of the document, whereas, don't change other portion of the document. It reserves all features of the prototype of the document.

CLAIMS

What claimed is:

1. A system of computer implementation for inserting hierarchical data into an existed target document, the system comprising: a module to connect a database and retrieve data from a database or another data source; a module to stored data in a multiple dimensional data repository and provide a search or retrieve path complying with regular expression; a module to generate a hierarchical document model based on scanning the target document including using directives as hint to create sub-model; a module to recursively transverse and insert the elements of the document model with the retrieved data.

2. The system of inserting hierarchical data into an existed document of claim 1, the database connection and retrieve module uses the XML script file to provide the connection parameters and query script. Alternatively, the module can be integrated into another system via the application programming interface to connect databases. An user can also employs other industrial standard modules to connect a database and retrieve data from the databases or from other data sources.

3. The system for inserting hierarchical data into an existed document of claim 1, wherein a data store module is to store data retrieved from databases or other data sources into a predefined multiple dimensional data repository. The data repository is a compound data structure to store data and provides search path complying with regular search expression. The data repository can be made as persistent object or as run-time object residing in computer memory.

4. The system of inserting hierarchical data into an existed document of claim 1, wherein the generation module of a document model is a process for the system to create a hierarchical document model based on scanned-in the existed document and on the using directives embedded in document. The document model is a derivation of generic data structure to store the elements of the target document elements. The system uses these directives as hints to perform the generation of both document and sub-document models. These directives can be nested, users can insert sub-directives between existed directives to build a hierarchical document model. These directives can be represented by, not restrict to, a set of iteration tags such as pairs of start iteration and end iteration tags. These iteration tags instruct the system to transversely iterate the contents between start and end iteration tags.

5. The system of inserting hierarchical data into an existed document of claim 1, wherein a data insertion module is a recursive process for the system to insert data into the created document model. The process transversely iterates the elements of the document model and parses these elements using the system defined expression. It then inserts the retrieved correspond data to the appropriate elements of the document model.

6. The system of inserting hierarchical data into an existed document of claim 1, wherein the existed document is a textural or text contained document in computer readable format.

INVESTOR IN PEOPLE

Application No: GB 0202848.8 **Examiner:** Ben Widdows
Claims searched: 1-6 **Date of search:** 10 January 2003

Patents Act 1977 : Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance	
X	at least 1,3&6	US 2001/0052908 A	(HARTMAN) see abstract and paragraph [0066] on page 6
X	at least 1,3&6	US 6282539 B1	(LUCA) see summary of invention
A	-	US 5539903 A	(IBM) see abstract
A	-	EP 1122692 A2	(SOLIDWORKS CORP) see abstract

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC[V]:

G4A

Worldwide search of patent documents classified in the following areas of the IPC[7]:

G06F

The following online and other databases have been used in the preparation of this search report:

WPI, EPODOC, JAPIO
THE INTERNET